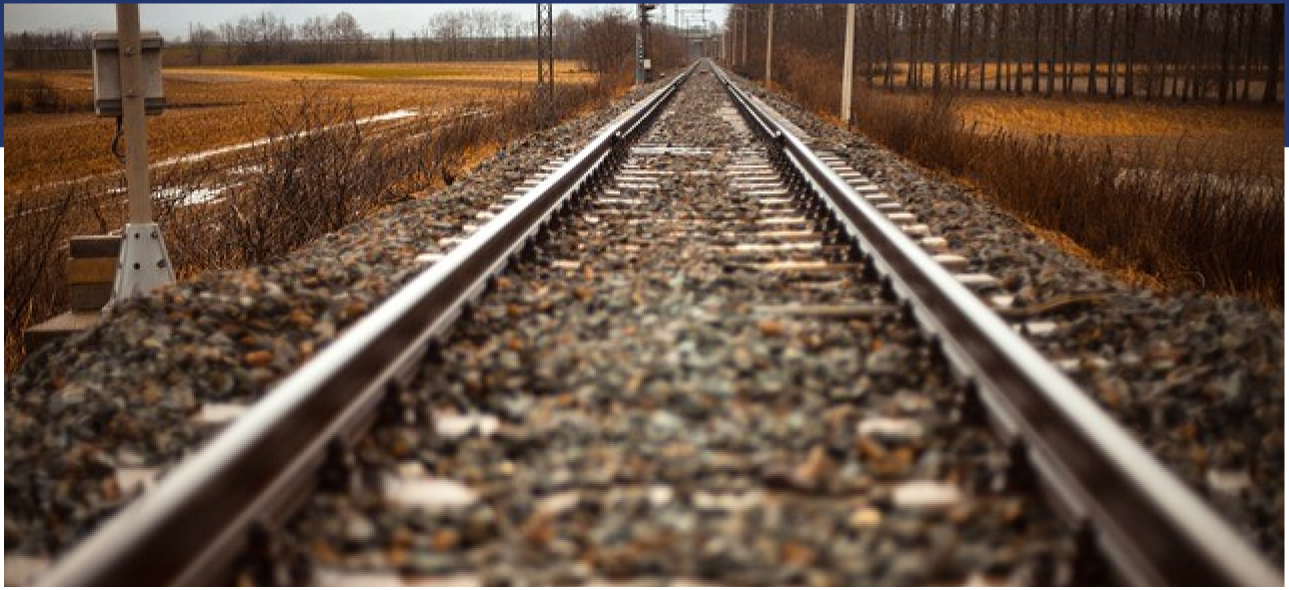
Parallel programming

Introduction



FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Why should you care about it?

- Sometimes you **want to obtain results faster**
  - an algorithm with time-consuming computations
  - a large amount of data

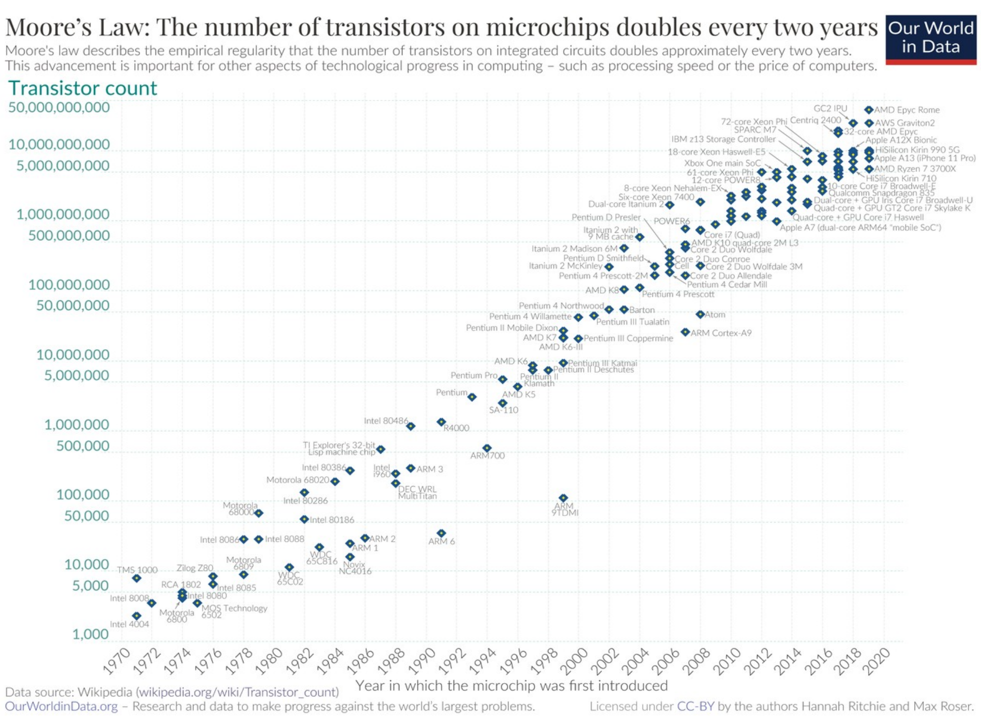
Applications: scientific computing (*simulations, calculations*), big data computing (*faster processing, databases*), machine learning, deep learning

- Sometimes you **have limited time to fulfill a task**
  - sequential execution is too slow
  - real-time processing

Benefit: some general principles apply to thinking about the architecture of separate programs for related tasks

Why should you care about it?

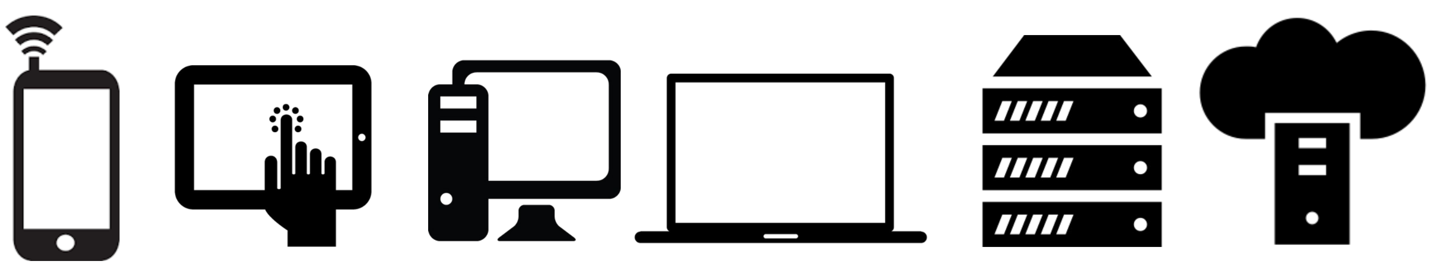
- Parallel computing is a dominant player in scientific and cluster computing. Why?
- Moore’s law (number of transistors doubles about every 2 years; for the same price, price per power halving) is reaching its limits
- Increase in transistor density is limited
- Memory access time has not been reduced at a rate comparable with processing speed



2. This chart shows the Linley Group’s “Cost Per Transistor” curve (2017) taken from Cadence’s “Breakfast Bytes” blog.

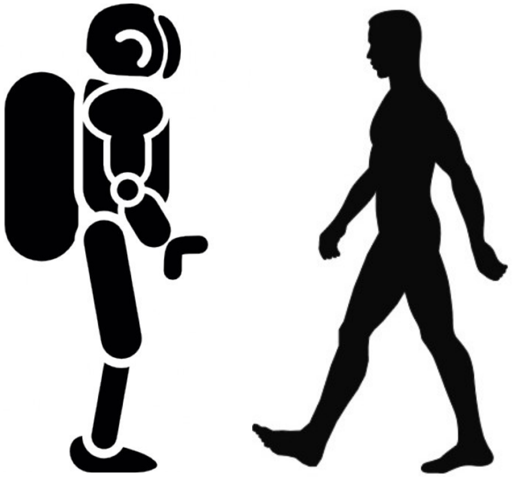
Why should you care about it?

- How to get out of this trap?
  - The most promising approach is to have multiple cores on a single processor
  - The number of cores is increasing, speed per core is improving more slowly
  - Today's desktop computers (2025)
    Intel Core Ultra 9 285 - 24 cores (16 efficient, 8 performance), 24 threads, 2.5 GHz performance, 1.9 GHz efficient, TDP 182W, Boost 5.6 GHz.
    AMD Ryzen 9 9950X3D - 16 cores, 32 threads, 4.3GHz, TDP 170W, Boost 5.7 GHz
  - Parallel computing can be found in many devices today:



OK; however, it should be a task for the compiler and not for me!!!

- Yes, the compiler can help you, but without your guidance, it is not able to make it all the way to a successful result.
  - Parallel programs often look very different than sequential ones
  - An efficient parallel implementation of a serial program may not be obtained by simply parallelizing each step
  - Rather, the best parallelization may be obtained by stepping back and devising an entirely new algorithm
  - Instruction level parallelization



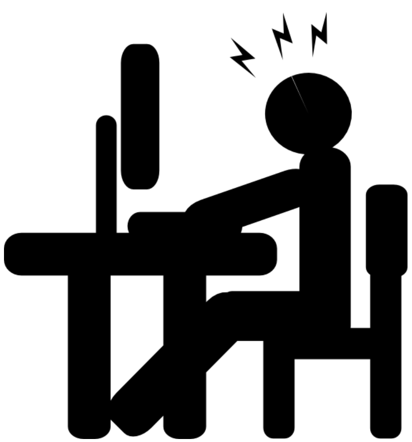
What is the aim of labs?

- To get a feel for parallel programming

  1) Understand what makes parallelization **complicated**
  2) Which **problems** can occur during parallelization
  3) What can be a **bottleneck**
  4) How to think about **algorithms** from the parallelization point of view

  Familiar terms: race condition, false sharing, synchronization, deadlocks, communication overhead, work imbalance, idling, alternative algorithm design vs. sequential version

- To get basic skills in common parallel programming frameworks
  1) for multi-core processors
  2) for computer clusters
  3) for GPU (nice opportunity to play with)



Seminar topics

- OpenMP – for multi-core processors, an easy way to parallelize originally sequential code, UMA concept
- MPI – for computer clusters, concept of units communicating through messages, NUMA concept
- Numba – computation on GPU
- Theoretical seminars – help you prepare for the exam

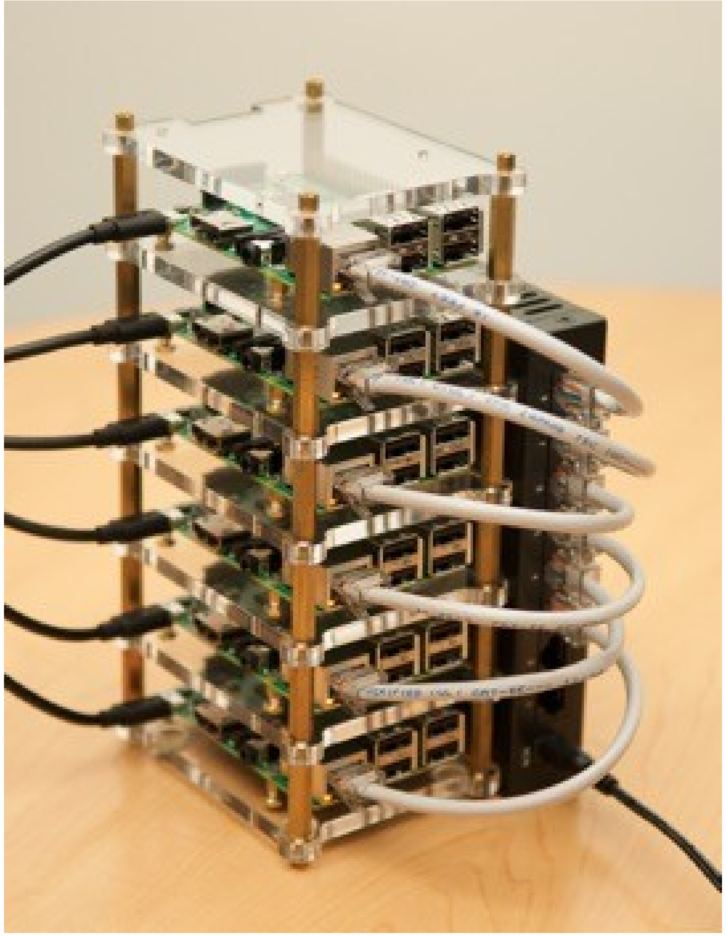
Basic terminology

- Processor: The physical chip that contains one or more cores.
- Core: A hardware execution unit inside the processor that can execute instructions independently.
- Thread: An execution context within a core, with its own program counter and registers, allowing a core to execute multiple instruction streams

Core != Thread (Simultaneous Multi-Threading [AMD], Hyper-Threading [Intel])

Using OpenMP vs MPI

AMD Threadripper – use OpenMP



Raspberry Pi stack – use MPI

Course web

- Course page https://cw.fel.cvut.cz/wiki/courses/pag/start
  - Detailed plan of labs, grading
- 5 homework assignments with strict deadlines during the semester

HW1	HW2	Report	HW3	HW4
7 points	8 points	7 points	11 points	7 points

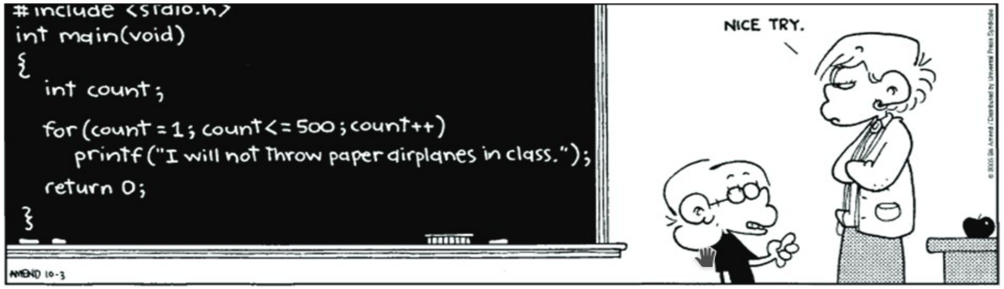
- Theoretical test in the 9th lecture (10 points)
- You need to obtain at least 25 points during the semester

Be ethical in homework solutions

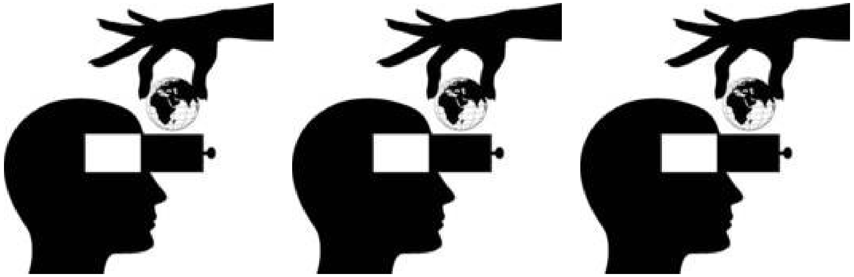
- Write your own code
- Do not plagiarise
- Adhere to the university rules on AI use

What does this course require?

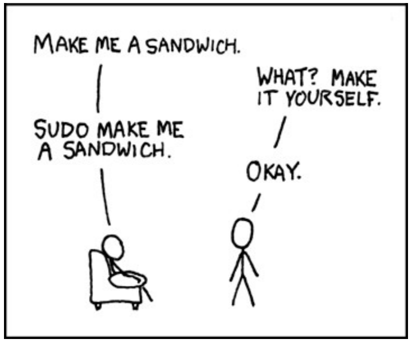
- Knowledge of C, C++ and basics of Python



- Analytical thinking and being open-minded

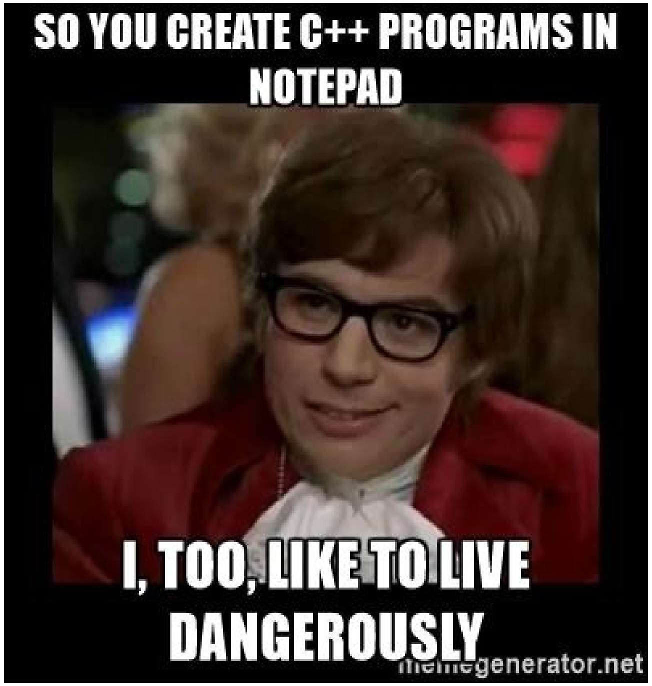


- Basic skills with Linux – shell, ssh, etc. (for MetaCentrum)



Setting up the environment

- Installation during the lab is recommended to detect problems
- Be prepared for coding next week
- Small HelloWorld examples are prepared for you to check if your environment runs smoothly
- Recommendations follow



Preferred developer tools

- Linux, Mac OS, Windows
  - CMake and g++
  - IDE: **CLion**
    - https://download.cvut.cz JetBrains
  - Libraries for support of OpenMP and MPI
  - Lab and homework skeletons are provided as CMake projects
  - See guide in the following slides, depends on your platform
- Windows+Visual Studio? :(
  - Use at your own risk
  - **Do not use MSVC** (no support for newer OpenMP)

1. IDE and codes

- Download and install CLion
  - https://download.cvut.cz/jetbrains/, free educational license
  - https://www.jetbrains.com/clion/

- Download `introduction_helloworlds.zip` skeleton and unzip it in your file system
  - https://cw.fel.cvut.cz/wiki/courses/pag/cviceni

introduction_helloworlds.zip

2. Managers and libraries installation

- This step varies for each platform!!
- We provide instructions for
  - Ubuntu apt manager (linux)
  - Windows with MSYS2 distribution manager (MINGW)
  - Windows with WSL (Windows Subsystem for Linux)
  - MacOs with Homebrew manager
- Keep the variant relevant for you

2. Ubuntu (linux)

- Install g++ and cmake

```
>> sudo apt install g++ cmake gdb
```

- GCC should contain built-in OpenMP support
- Install MPI library

```
>> sudo apt install libopenmpi-dev
```

2. Windows with MINGW

- Install msys2, see this link
- In the console MSYS2 MINGW32/64 do the following

```
>> pacman -Syu
>> pacman -Su
>> pacman -S base-devel mingw-w64-x86_64-toolchain
>> pacman -S mingw-w64-x86_64-msmpi
```

Library detail: https://packages.msys2.org/base/mingw-w64-msmpi

- Add msys2 directories to your PATH environment variable, e.g.,

Depends on your selected installation folder (just default location present)

```
C:\msys64
C:\msys64\mingw64\bin
```

How to add a path to the PATH variable e.g. this guide

2. Windows with WSL

- WSL already installed?
  - Check WSL distributions in Powershell
    - See with `wsl -l -v`, Name Ubuntu should be listed and displayed with * before name (meaning default distribution)
    - To add Ubuntu distribution run: `wsl --install - d Ubuntu`
    - To set Ubuntu as default distribution run: `wsl -s Ubuntu`
  - Ensure access to Ubuntu terminal:
    - Directly from Powershell with `C:\Windows\system32\wsl.exe -d Ubuntu`
    - Installed Ubuntu terminal as an application in Windows
    - As one of possible terminals inside Windows terminal this link
- Pure installation?
  - Install WSL, see this link
    - In Powershell run: `wsl –install`
  - Install Ubuntu distribution via microsoft store if not already installed, see this link

2. Windows with WSL

- Open Ubuntu terminal, initiate system if first run (user access setup), then run the following

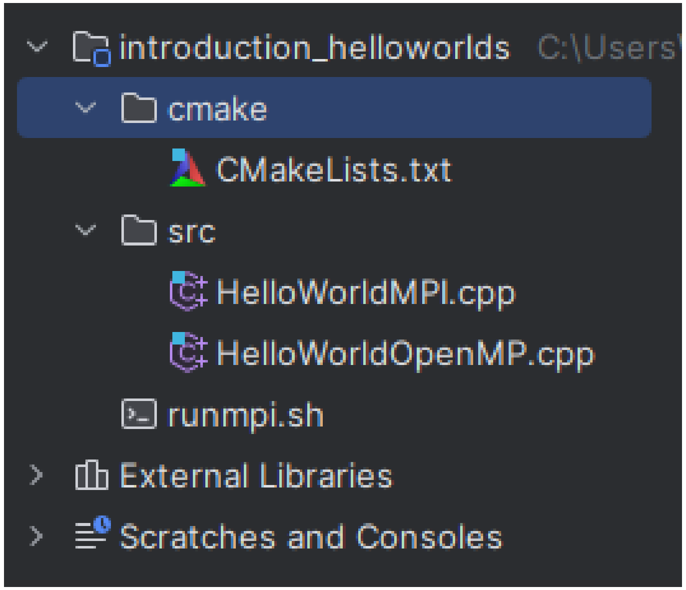
```
>> sudo apt update

>> sudo apt install g++ cmake gdb

>> sudo apt install libopenmpi-dev
```

2. MacOS

- Install package manager Homebrew (if you don't have it)
- Install g++ and cmake by Homebrew
  `>> brew install gcc cmake`
- Install OpenMP and MPI
  `>> brew install libomp open-mpi`
- Find the installed gcc and g++ executable. Look for installed versions under location /opt/homebrew/bin, there should be programs `gcc-V` and `g++-V`, where V means your version
  - remember the installed version, you will need it later

3. Open skeletons with CLion

- You need to open the project properly!
- Project should contain `cmake` and src directories
- CLion will try to load the CMake project, ignore that for now, continue to the next step

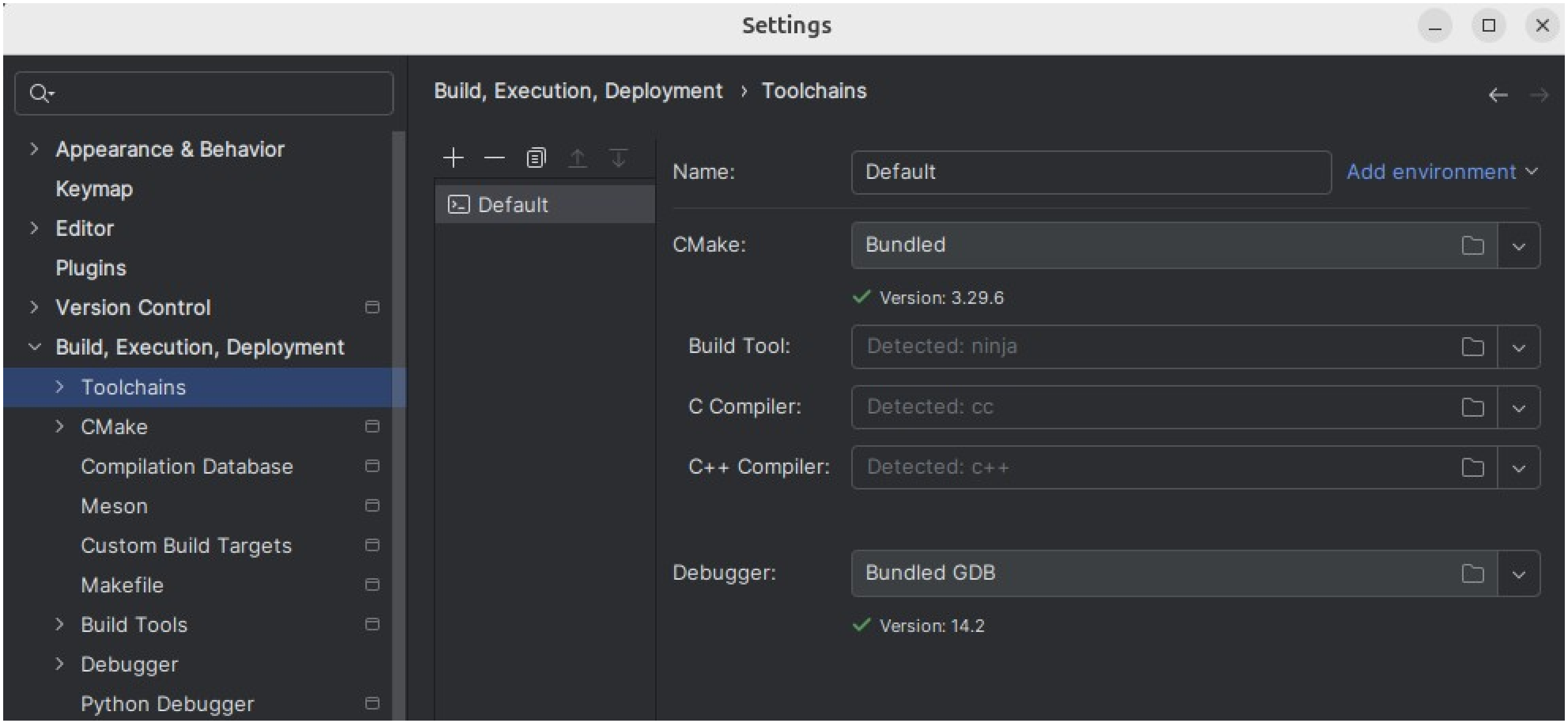


4. CLion Create toolchain

- Set up toolchain to build and run from CLion
- Use the variant relevant for you in this step

4. Ubuntu Toolchains settings

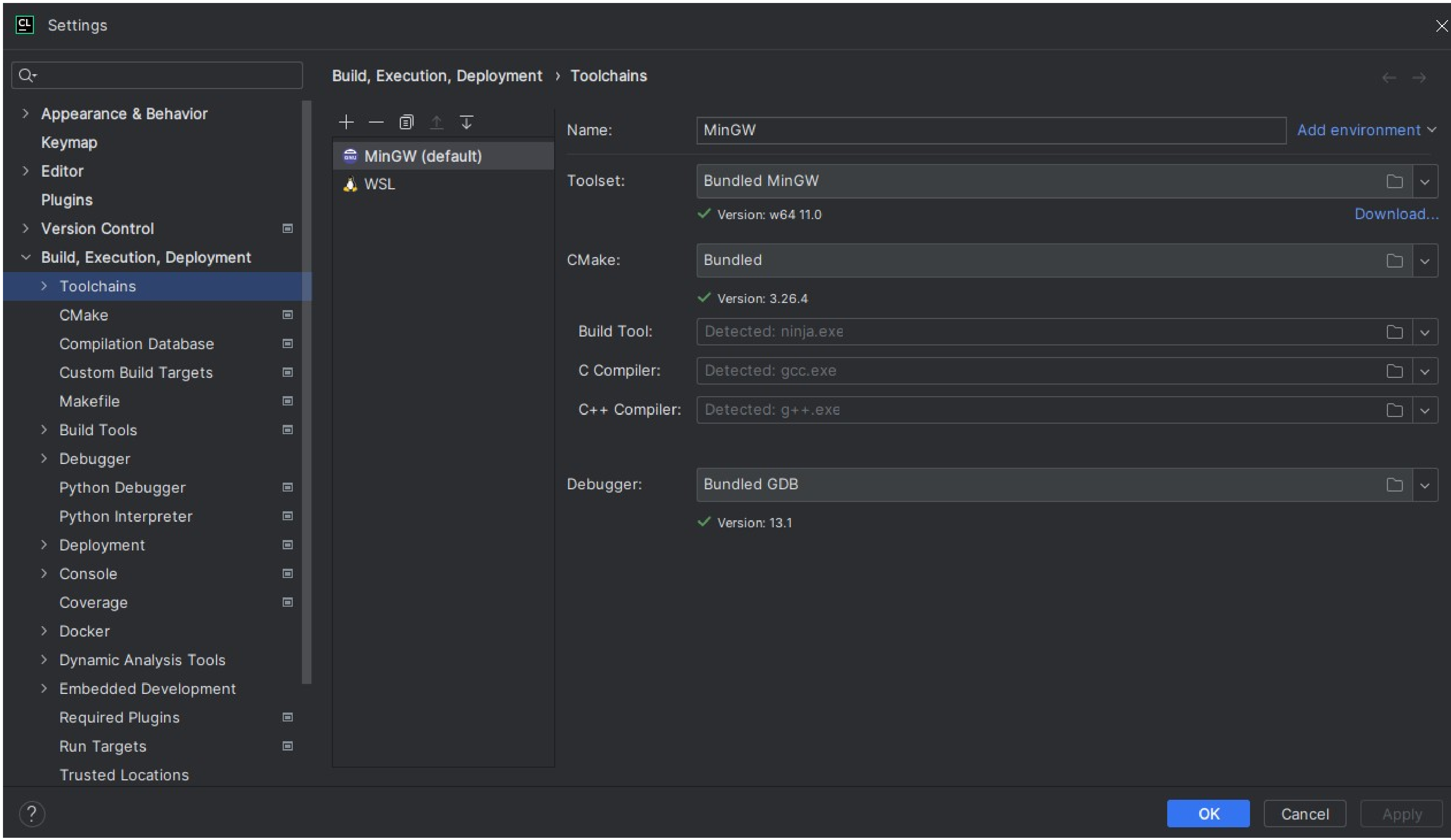
- All should detect automatically



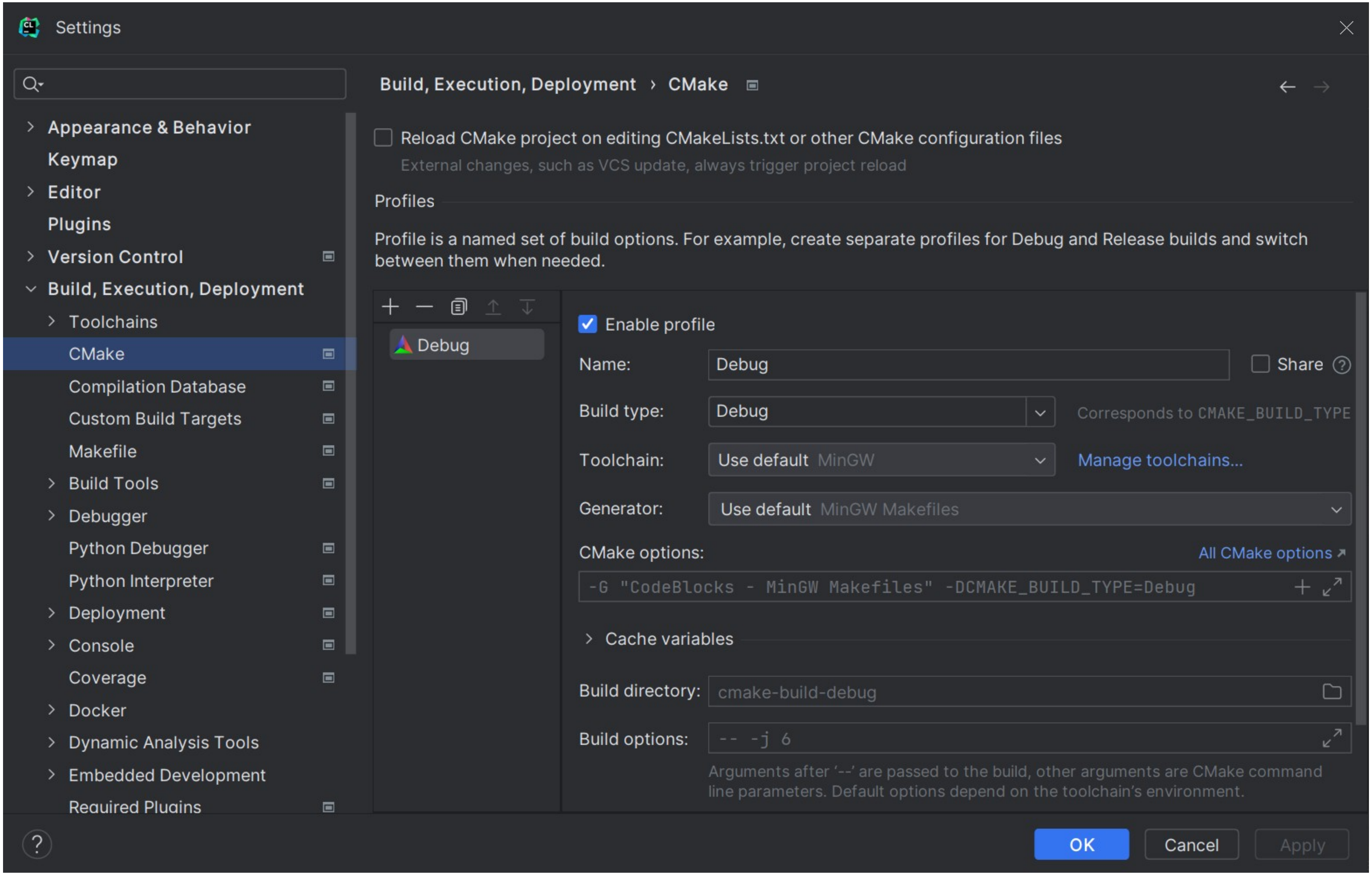
4. Windows MINGW

- Create MinGW toolchain in CLion, see this link.
- If msys2 is installed in default location, set `C:\msys64\mingw64` as your MinGW Environment path (everything else should be detected automatically)
- Set up in CLion Settings -> CMake -> Generator on value "MINGW Makefiles"

4. Windows MINGW Toolchains settings



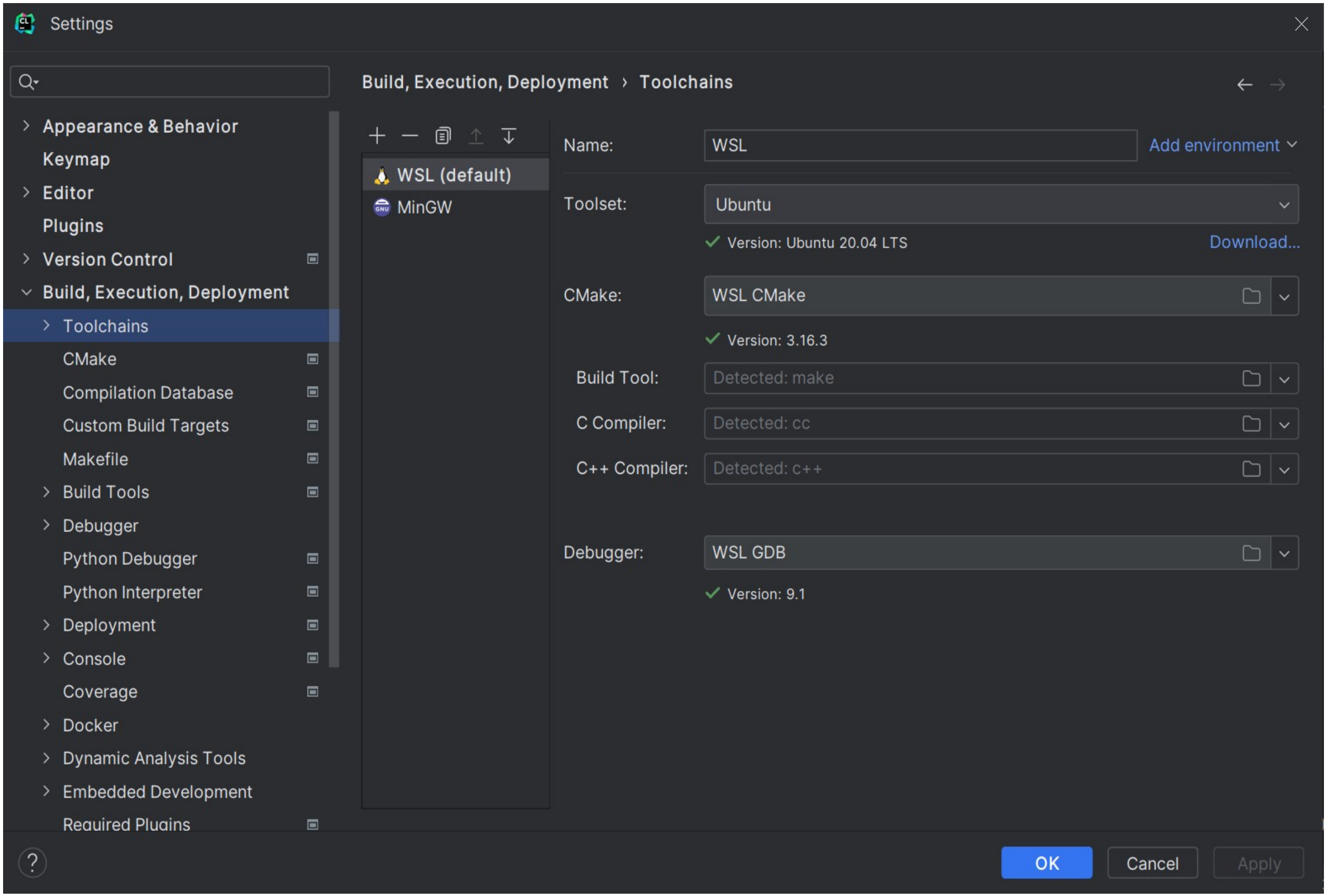
4. Windows MINGW CMake settings



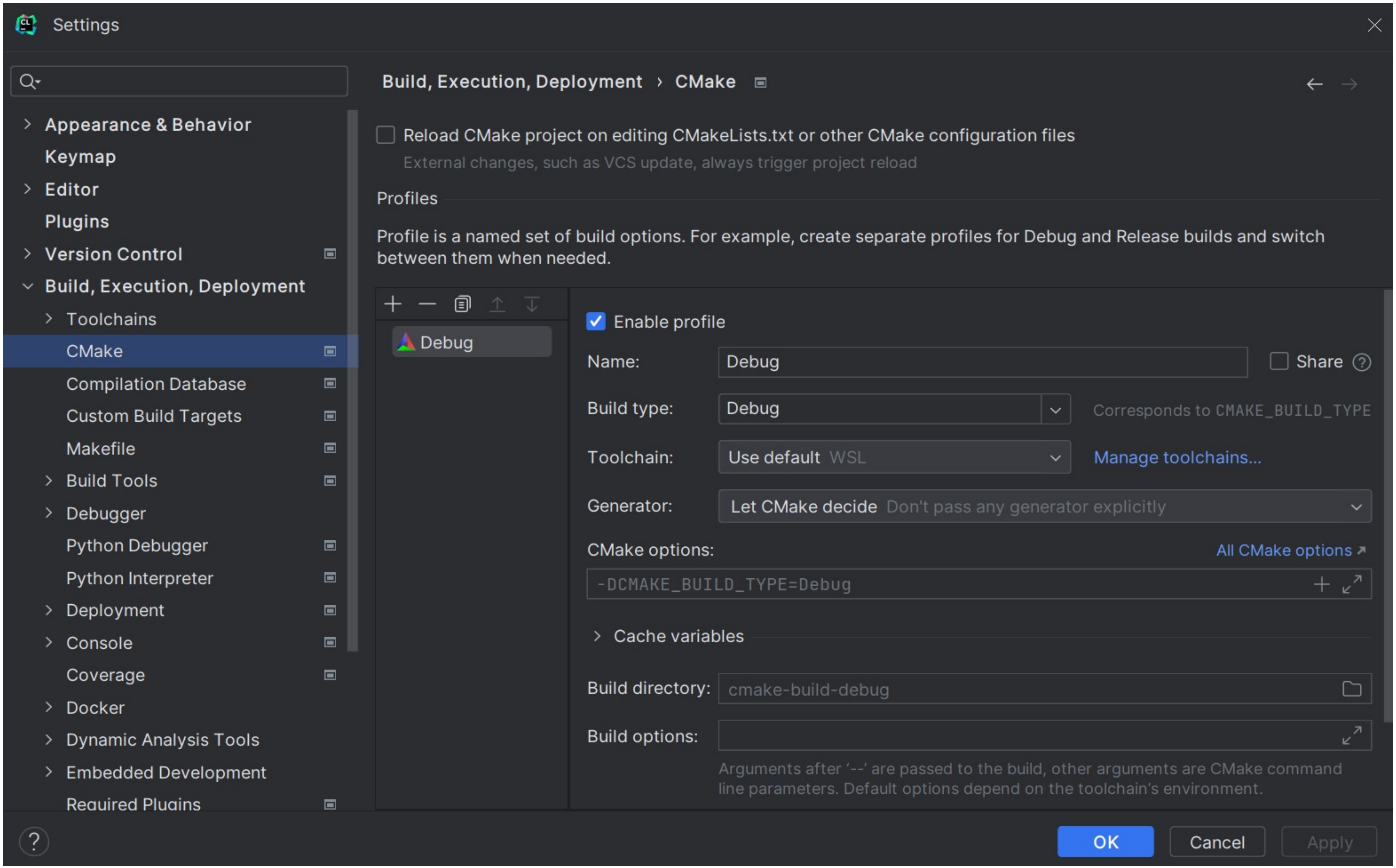
4. Windows WSL

- Set up WSL in CLion toolchains, see this link
- You can check your settings using screenshots in the following slides

4. Windows WSL Toolchains settings



4. Windows WSL CMake settings

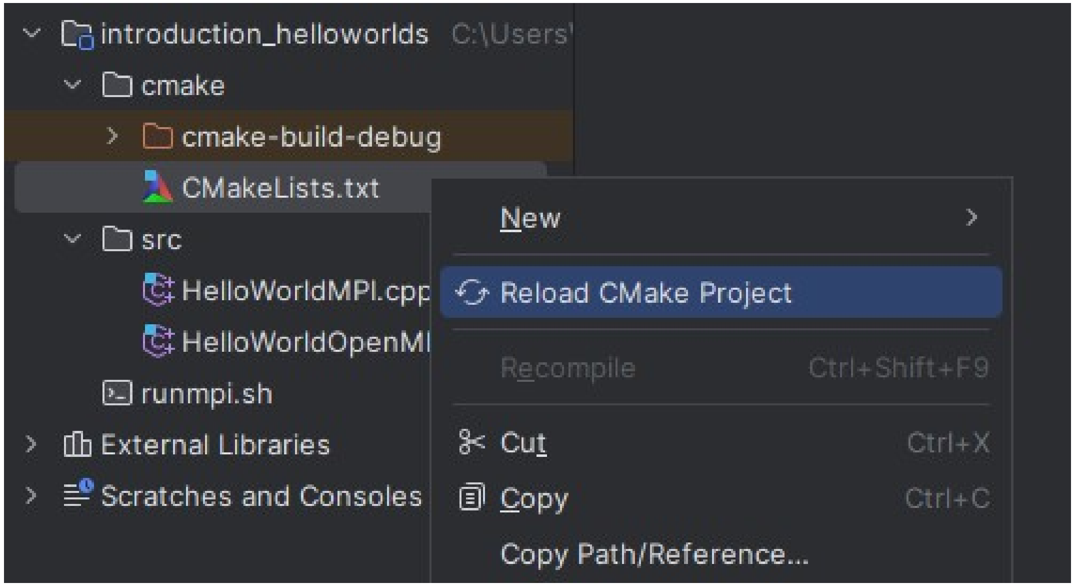


4. MacOS

- You should remember the installed gcc and g++ version
- Set C and C++ compilers in CLion: Settings → Build, Execution, Deployment → Toolchains → C compiler and C++ compiler (Compilers under Homebrew can be detected automatically, if not continue)
- Set manually paths to the compilers
  E.g. for version 14, specific paths will look like
  `C compiler: /opt/homebrew/bin/gcc-14`
  `C++ compiler: /opt/homebrew/bin/g++-14`

5. Reload CMake

- Delete generated folder `cmake-build-debug`
- Right-click the cmake folder and select `Reload CMake Project`



6. Check CMake console output

- You should see in the CMake console output:
  Found OpenMP: TRUE
  Found MPI: TRUE

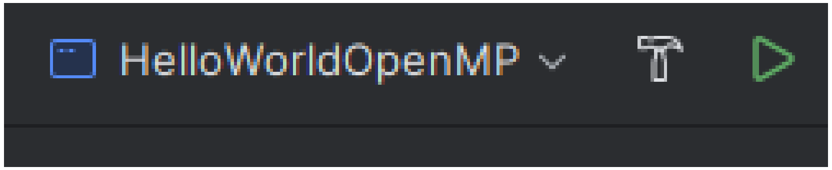
CMake   Debug

```
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Found MPI_C: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so (found version "3.1")
-- Found MPI_CXX: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi_cxx.so (found version "3.1")
-- Found MPI: TRUE (found version "3.1")
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/c/Users/stejs/Desktop/week1_codes/cmake/cmake-build-debug

[Finished]
```

OPENMP (needed for next week), MPI (there is time to solve issues, used later)

7. Run OpenMP example

- Select HelloWorldOpenMP
- Build with hammer icon
- Run with play button
- Output should look like



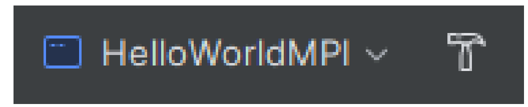

```
Number of available threads 4
This is thread 1 speaking
This is thread 0 speaking
This is thread 2 speaking
This is thread 3 speaking
Parallel block finished
Value of x: 550

Process finished with exit code 0
```

8. Run MPI example with mpiexec

- To run the MPI example, you need to run the compiled program with mpiexec utility
- For Windows with MinGW extra installation of mpiexec is needed:
  - Download MPI SDK containing mpiexec program (download msmpisetup.exe installer)
  - You can check whether Mpiexec.exe is installed in C:\Program Files\MPI\bin
  - Check the path listed under the PATH environment variable; if it is not found, add the path C:\Program Files\MPI\bin

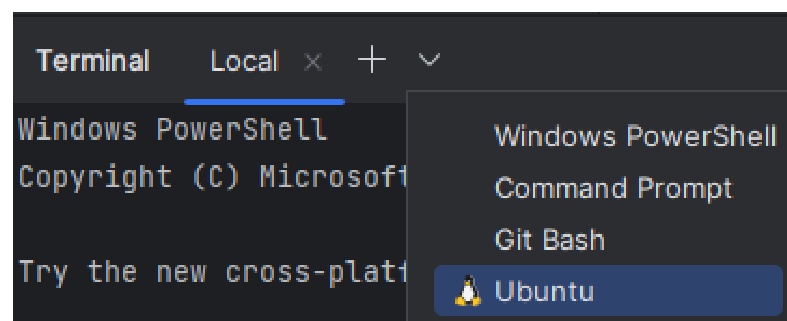
8. Run MPI example with mpiexec

- Select HelloWorldMPI
- Build with hammer icon



- Open console in CLion (For WSL usage you have to select Ubuntu console)
- Run with mpiexec

  `mpiexec -np 4 cmake/cmake-build-debug/HelloWorldMPI.exe` (Windows MINGW)
  `mpiexec -np 4 cmake/cmake-build-debug/HelloWorldMPI` (Windows WSL, Ubuntu, MacOS)



- Output should look like

  My ranking hello world example: 0
  My ranking hello world example: 1
  Total number of processes: 4
  My ranking hello world example: 3
  My ranking hello world example: 2

  Process finished with exit code 0